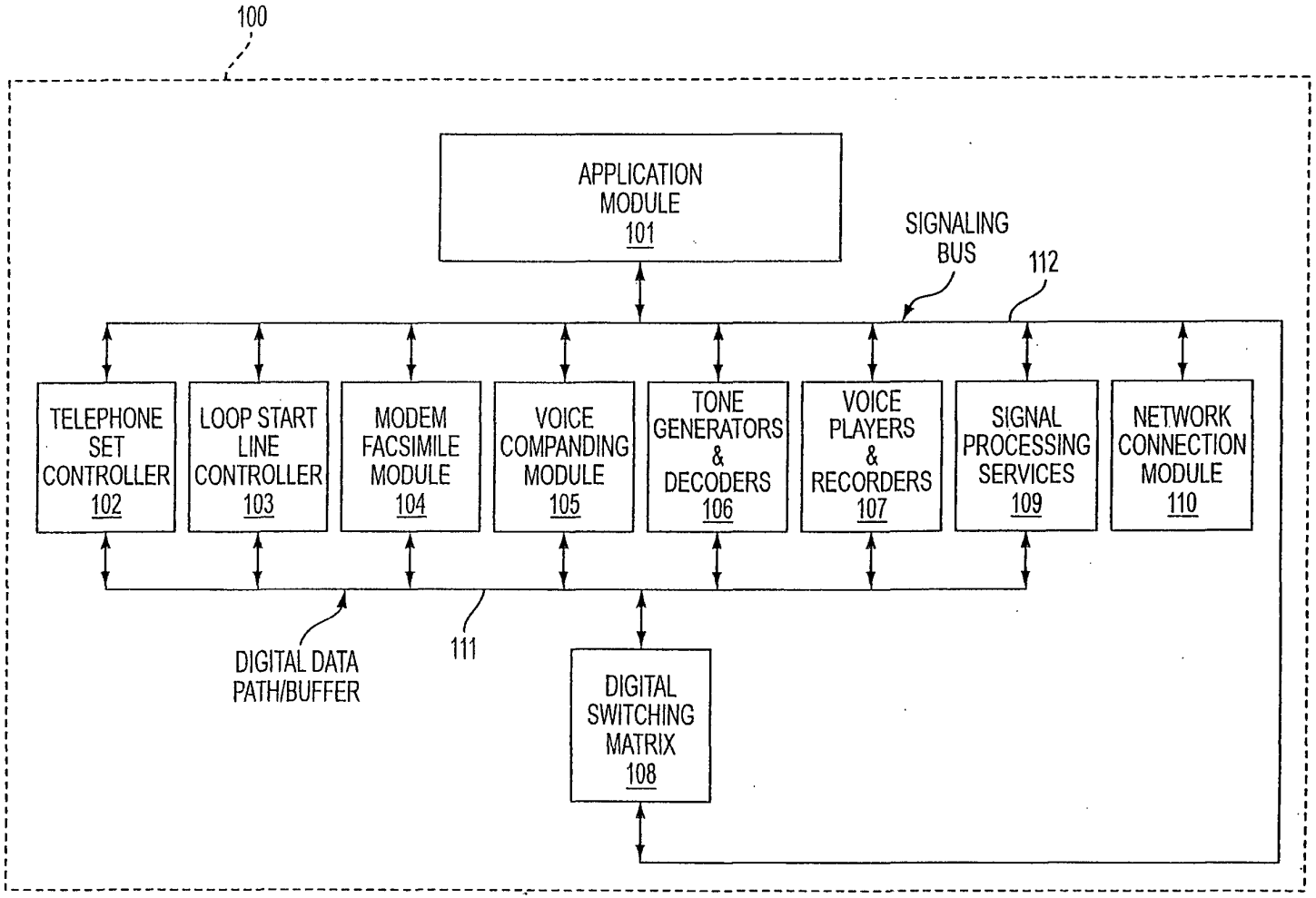
100
APPLICATION MODULE 101
SIGNALING BUS
112
TELEPHONE SET CONTROLLER 102
LOOP START LINE CONTROLLER 103
MODEM FACSIMILE MODULE 104
VOICE COMPANDING MODULE 105
TONE GENERATORS & DECODERS 106
VOICE PLAYERS & RECORDERS 107
SIGNAL PROCESSING SERVICES 109
NETWORK CONNECTION MODULE 110
DIGITAL DATA PATH/BUFFER
111
DIGITAL SWITCHING MATRIX 108

FIG. 1

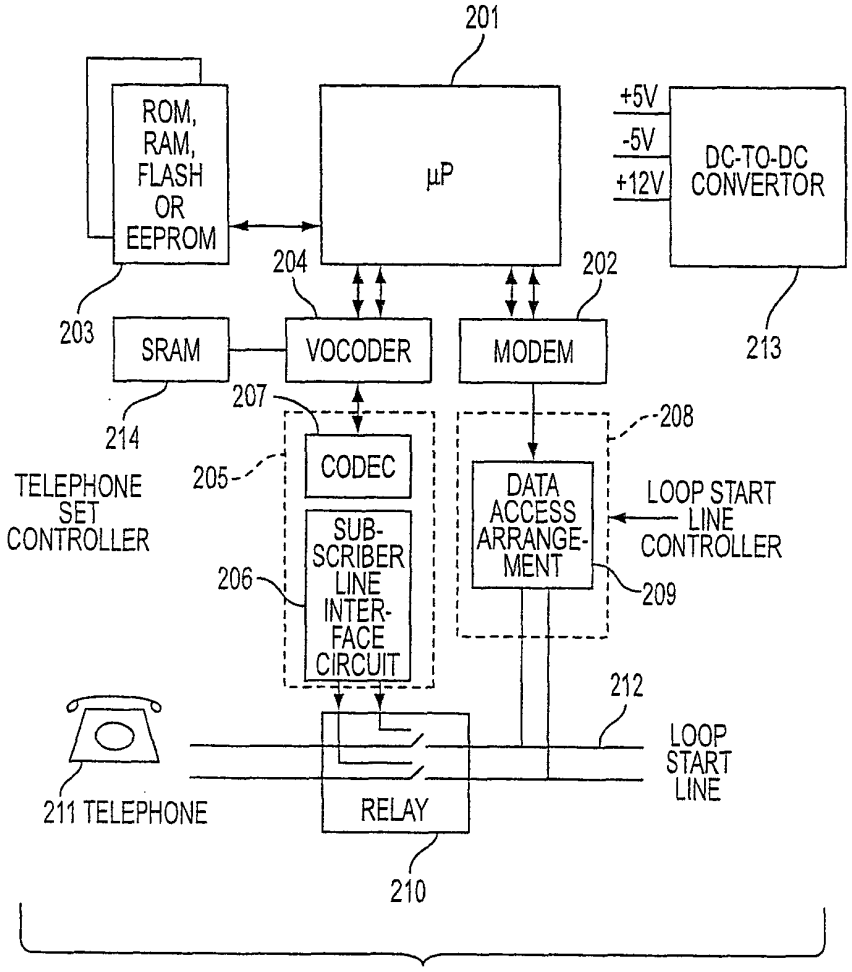
201
ROM, RAM, FLASH OR EEPROM
μP
+5V
-5V
+12V
DC-TO-DC CONVERTOR
203
204
202
213
SRAM
VOCODER
MODEM
214
207
208
TELEPHONE SET CONTROLLER
205
CODEC
DATA ACCESS ARRANGE-MENT
LOOP START LINE CONTROLLER
206
SUB-SCRIBER LINE INTER-FACE CIRCUIT
209
212
LOOP START LINE
211 TELEPHONE
RELAY
210

FIG. 2

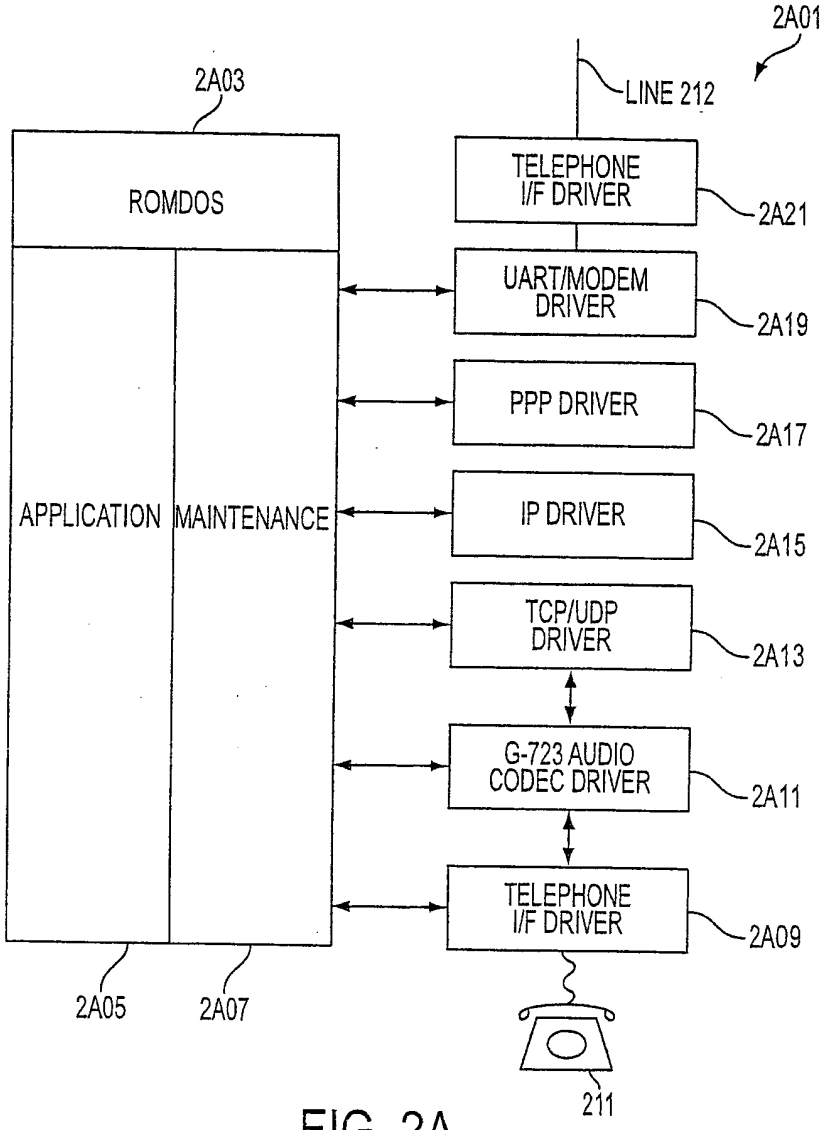
2A01
LINE 212
2A03
ROMDOS
APPLICATION
MAINTENANCE
TELEPHONE I/F DRIVER
2A21
UART/MODEM DRIVER
2A19
PPP DRIVER
2A17
IP DRIVER
2A15
TCP/UDP DRIVER
2A13
G-723 AUDIO CODEC DRIVER
2A11
TELEPHONE I/F DRIVER
2A09
2A05
2A07
211

FIG. 2A

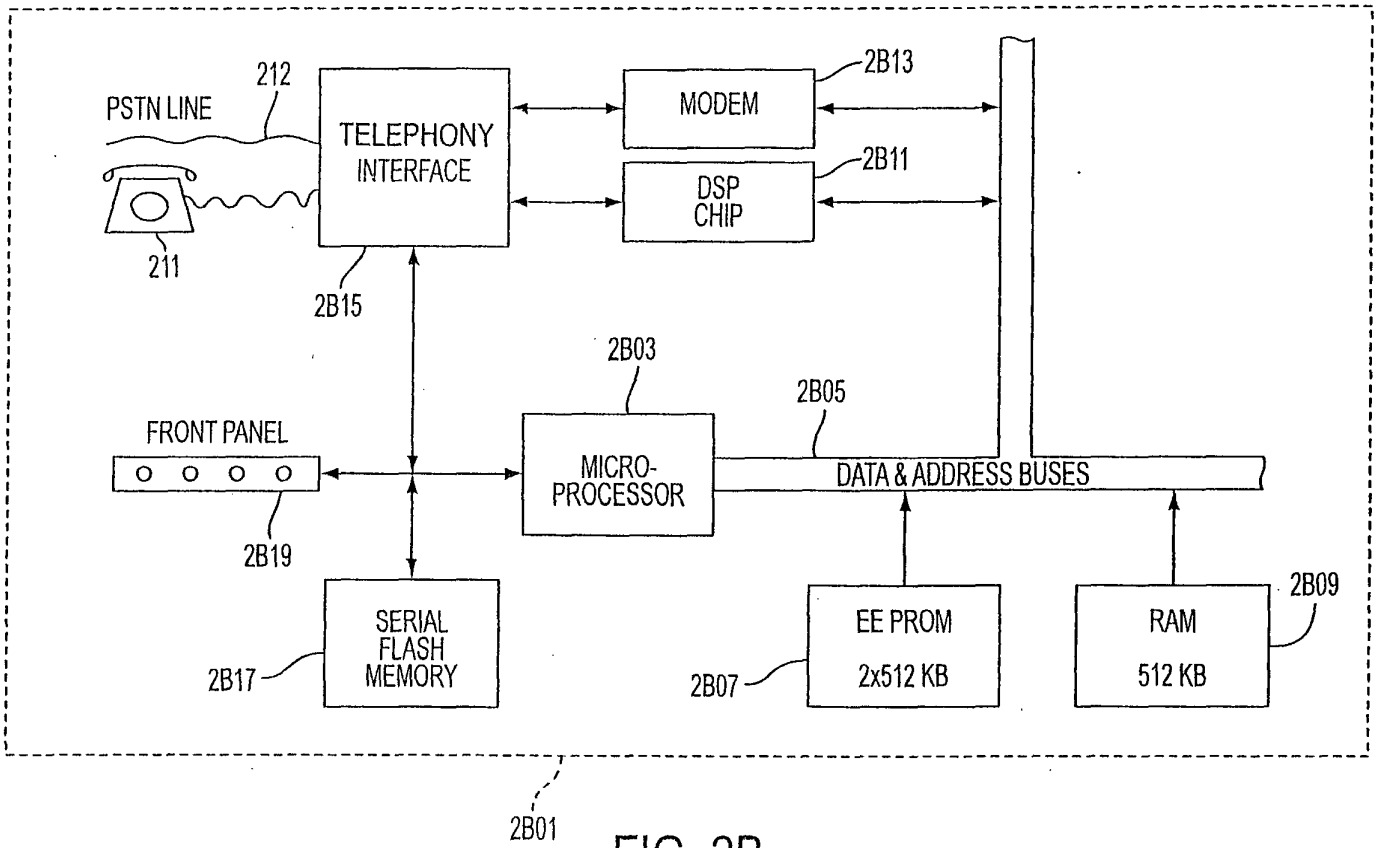
PSTN LINE
212
TELEPHONY INTERFACE
MODEM
2B13
DSP CHIP
2B11
211
2B15
2B03
2B05
FRONT PANEL
MICRO-PROCESSOR
DATA & ADDRESS BUSES
2B19
SERIAL FLASH MEMORY
2B17
EE PROM
2x512 KB
2B07
RAM
512 KB
2B09
2B01

FIG. 2B

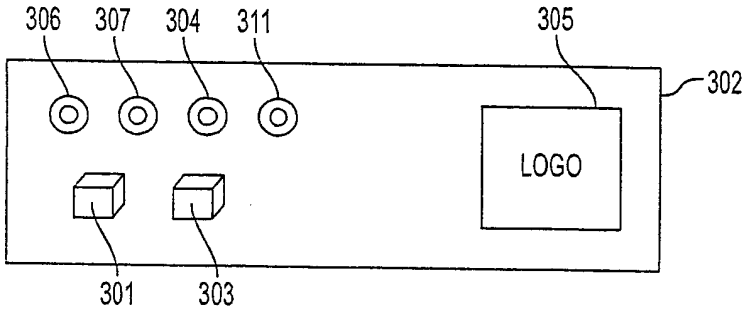

FIG. 3

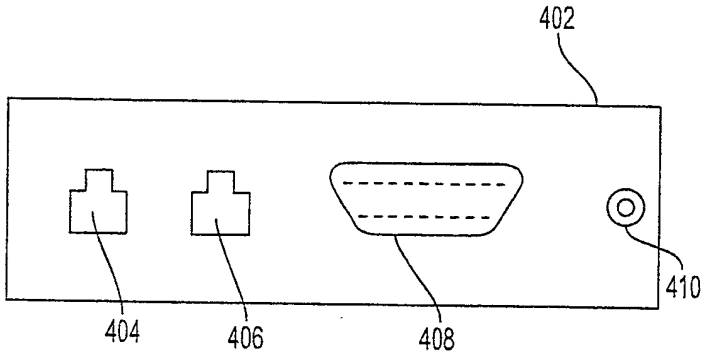

FIG. 4

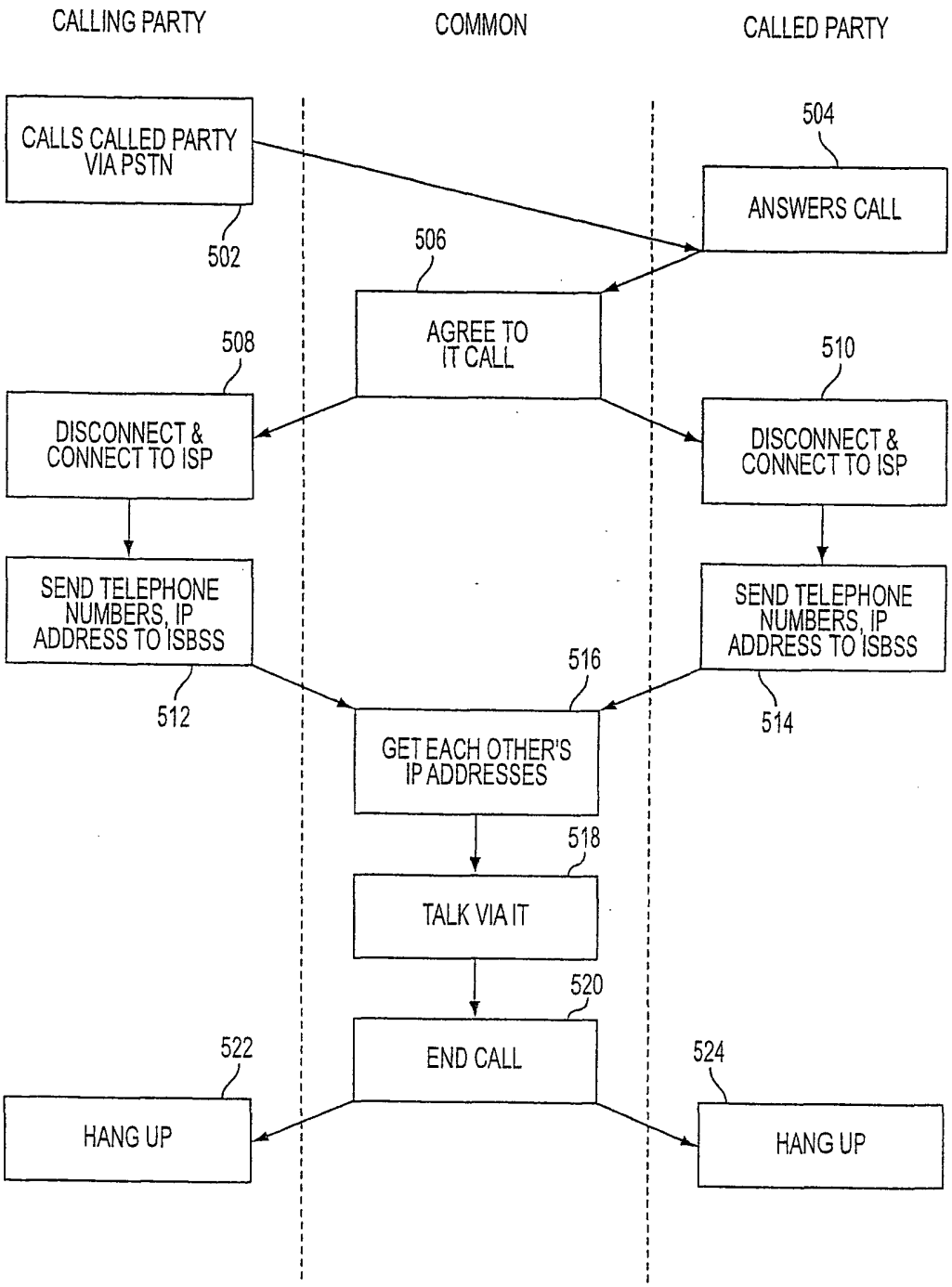
CALLING PARTY
COMMON
CALLED PARTY
CALLS CALLED PARTY VIA PSTN
502
504
ANSWERS CALL
506
AGREE TO IT CALL
508
DISCONNECT & CONNECT TO ISP
510
DISCONNECT & CONNECT TO ISP
SEND TELEPHONE NUMBERS, IP ADDRESS TO ISBSS
512
SEND TELEPHONE NUMBERS, IP ADDRESS TO ISBSS
514
516
GET EACH OTHER'S IP ADDRESSES
518
TALK VIA IT
520
END CALL
522
HANG UP
524
HANG UP

FIG. 5

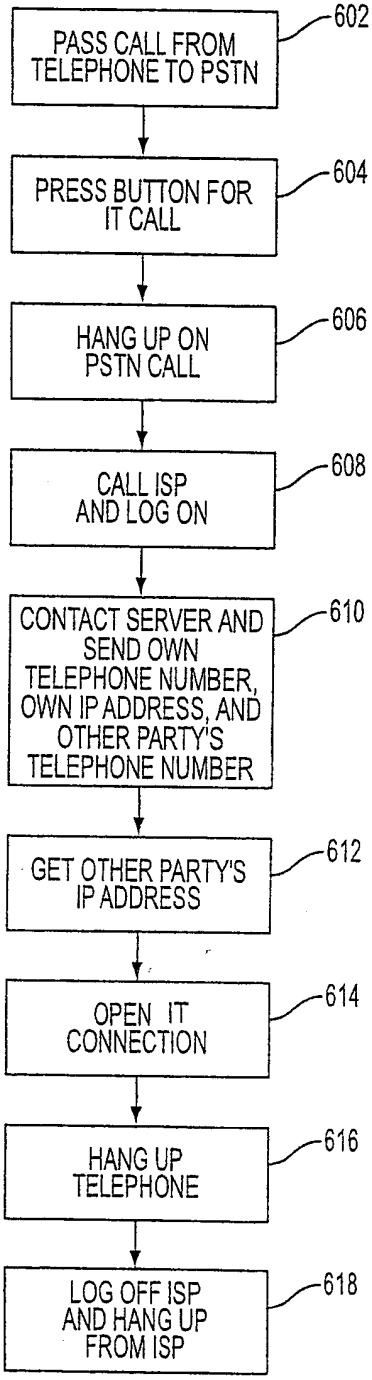
PASS CALL FROM TELEPHONE TO PSTN
602
PRESS BUTTON FOR IT CALL
604
HANG UP ON PSTN CALL
606
CALL ISP AND LOG ON
608
CONTACT SERVER AND SEND OWN TELEPHONE NUMBER, OWN IP ADDRESS, AND OTHER PARTY'S TELEPHONE NUMBER
610
GET OTHER PARTY'S IP ADDRESS
612
OPEN IT CONNECTION
614
HANG UP TELEPHONE
616
LOG OFF ISP AND HANG UP FROM ISP
618

FIG. 6

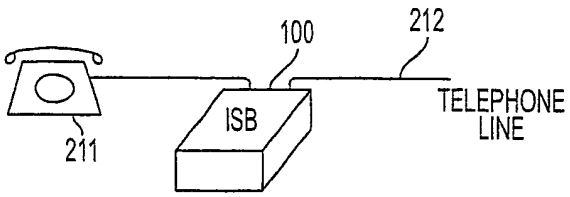

FIG. 7A

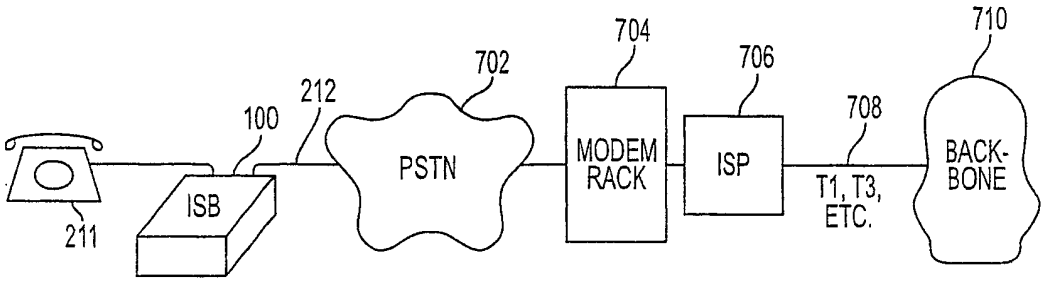

FIG. 7B

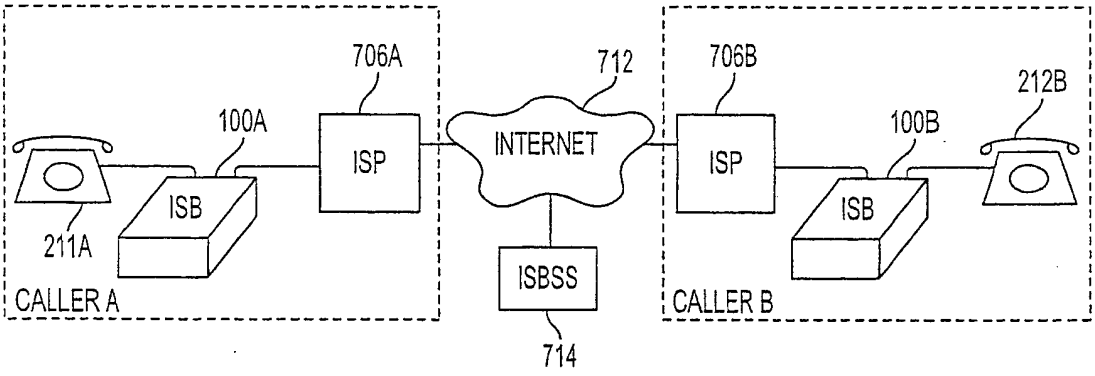

FIG. 7C

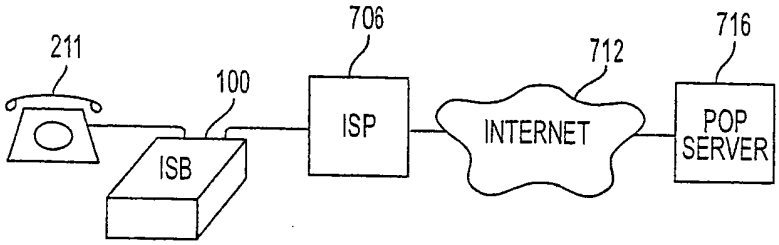
211
100
706
712
716
ISB
ISP
INTERNET
POP SERVER

FIG. 7D

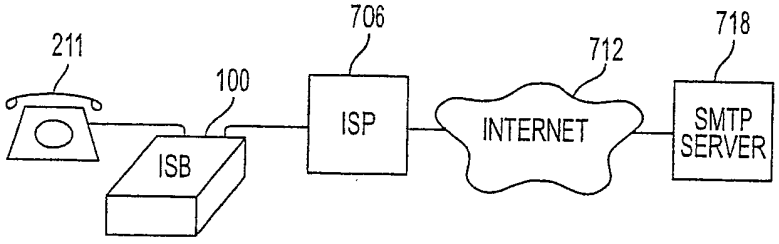
211
100
706
712
718
ISB
ISP
INTERNET
SMTP SERVER

FIG. 7E

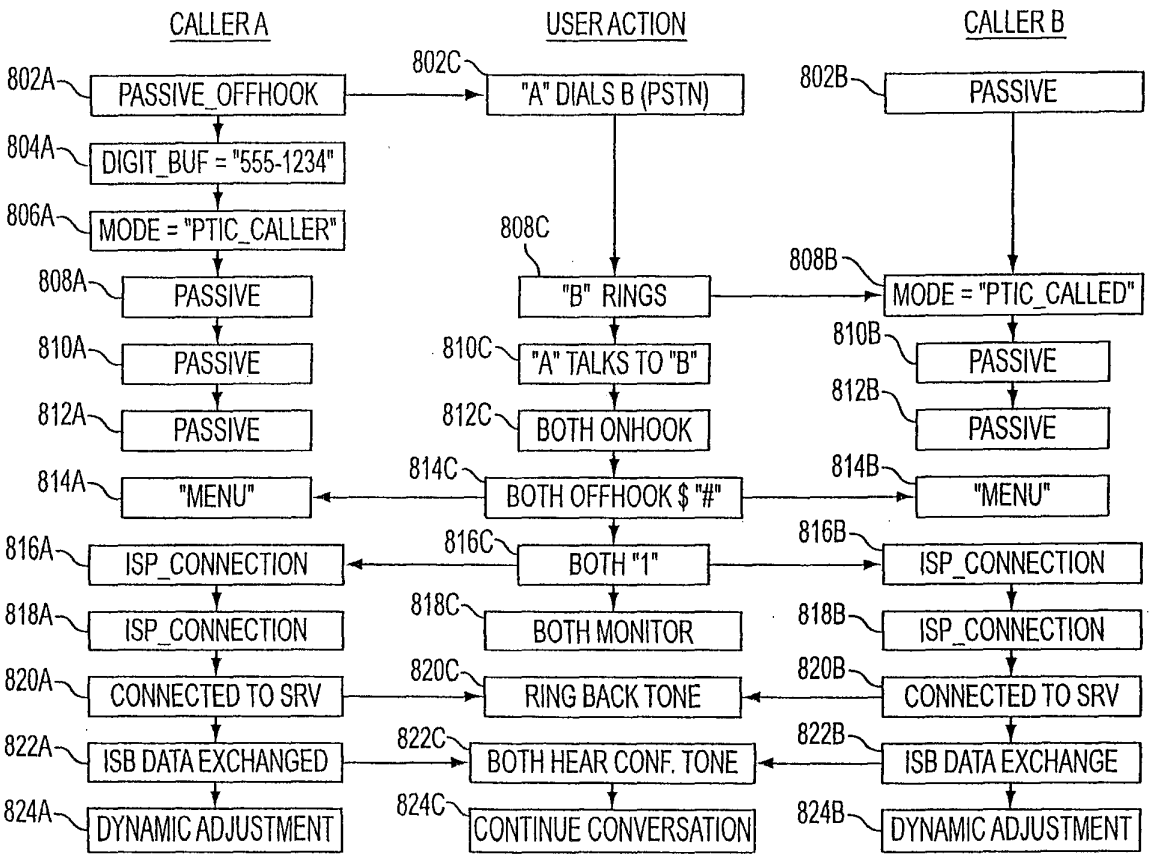
CALLER A
USER ACTION
CALLER B
802A
PASSIVE_OFFHOOK
802C
"A" DIALS B (PSTN)
802B
PASSIVE
804A
DIGIT_BUF = "555-1234"
806A
MODE = "PTIC_CALLER"
808C
808A
PASSIVE
"B" RINGS
808B
MODE = "PTIC_CALLED"
810A
PASSIVE
810C
"A" TALKS TO "B"
810B
PASSIVE
812A
PASSIVE
812C
BOTH ONHOOK
812B
PASSIVE
814A
"MENU"
814C
BOTH OFFHOOK $ "#"
814B
"MENU"
816A
ISP_CONNECTION
816C
BOTH "1"
816B
ISP_CONNECTION
818A
ISP_CONNECTION
818C
BOTH MONITOR
818B
ISP_CONNECTION
820A
CONNECTED TO SRV
820C
RING BACK TONE
820B
CONNECTED TO SRV
822A
ISB DATA EXCHANGED
822C
BOTH HEAR CONF. TONE
822B
ISB DATA EXCHANGE
824A
DYNAMIC ADJUSTMENT
824C
CONTINUE CONVERSATION
824B
DYNAMIC ADJUSTMENT

FIG. 8

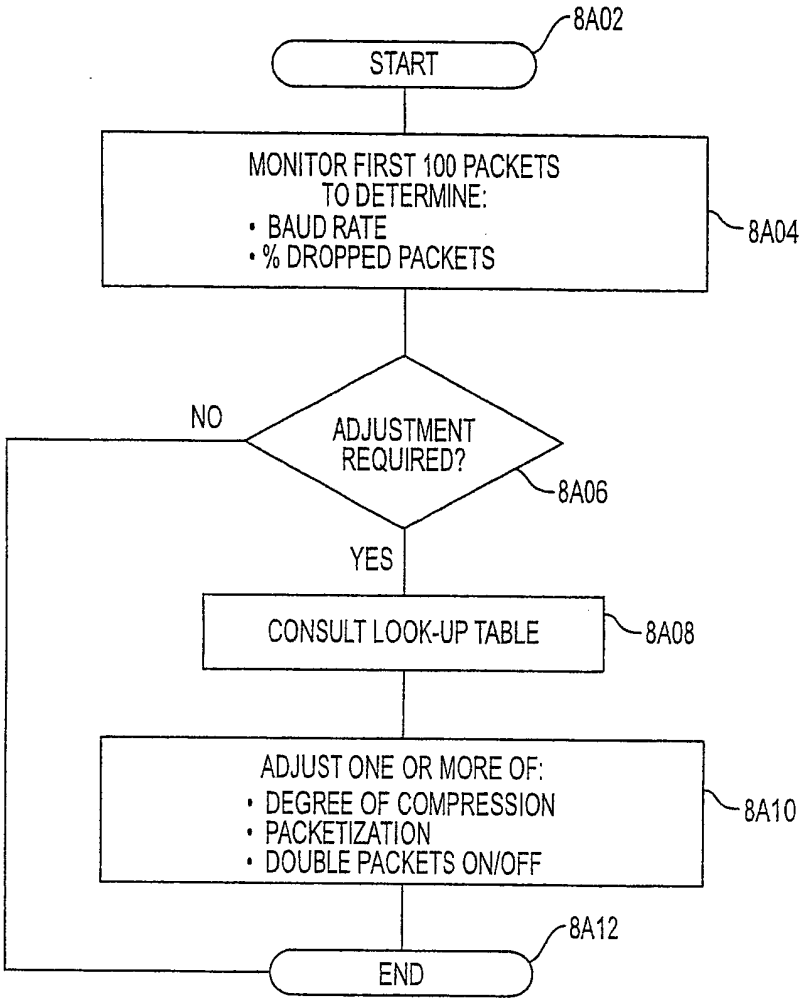
8A02
START
MONITOR FIRST 100 PACKETS TO DETERMINE:
• BAUD RATE
• % DROPPED PACKETS
8A04
ADJUSTMENT REQUIRED?
8A06
NO
YES
CONSULT LOOK-UP TABLE
8A08
ADJUST ONE OR MORE OF:
• DEGREE OF COMPRESSION
• PACKETIZATION
• DOUBLE PACKETS ON/OFF
8A10
8A12
END

FIG. 8A

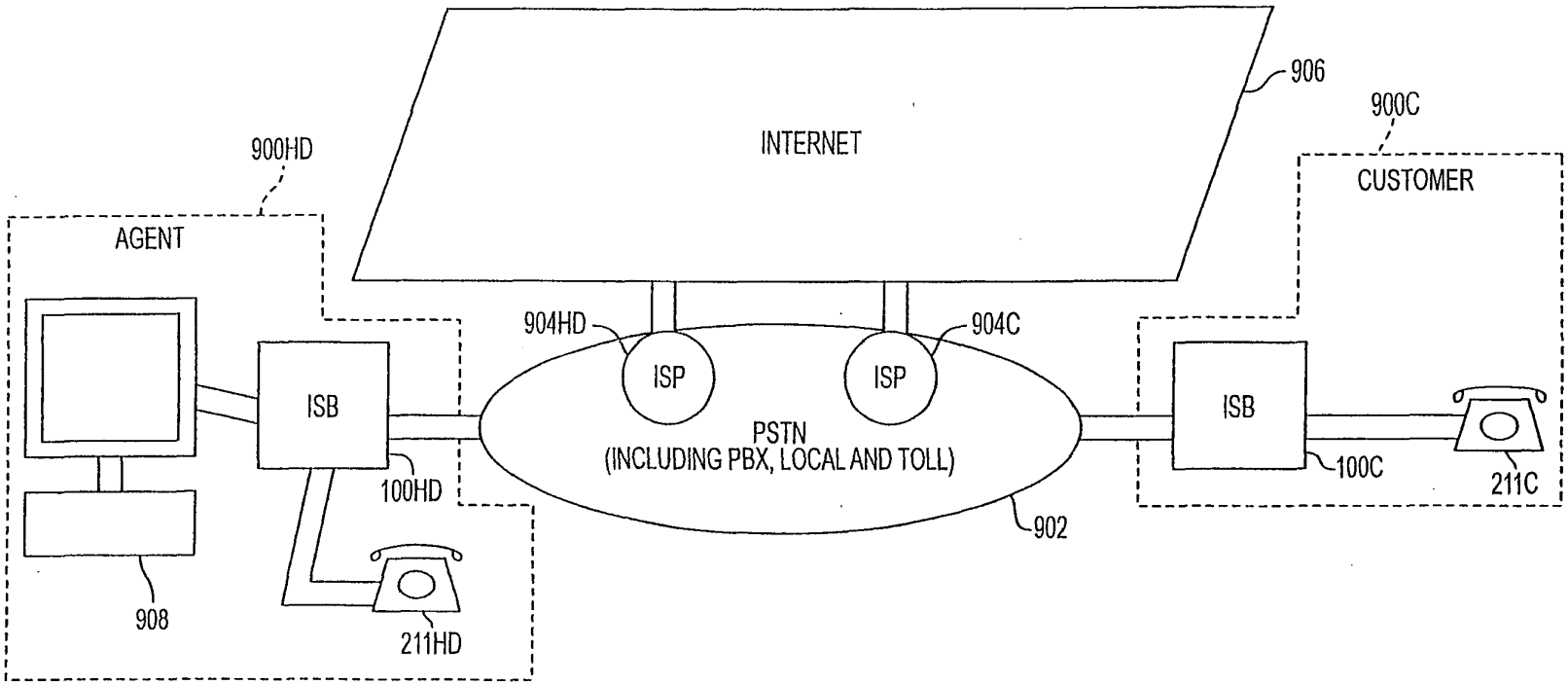
906
900C
INTERNET
CUSTOMER
900HD
AGENT
904HD
904C
ISP
ISP
ISB
ISB
PSTN
(INCLUDING PBX, LOCAL AND TOLL)
100HD
100C
211C
902
908
211HD

FIG. 9

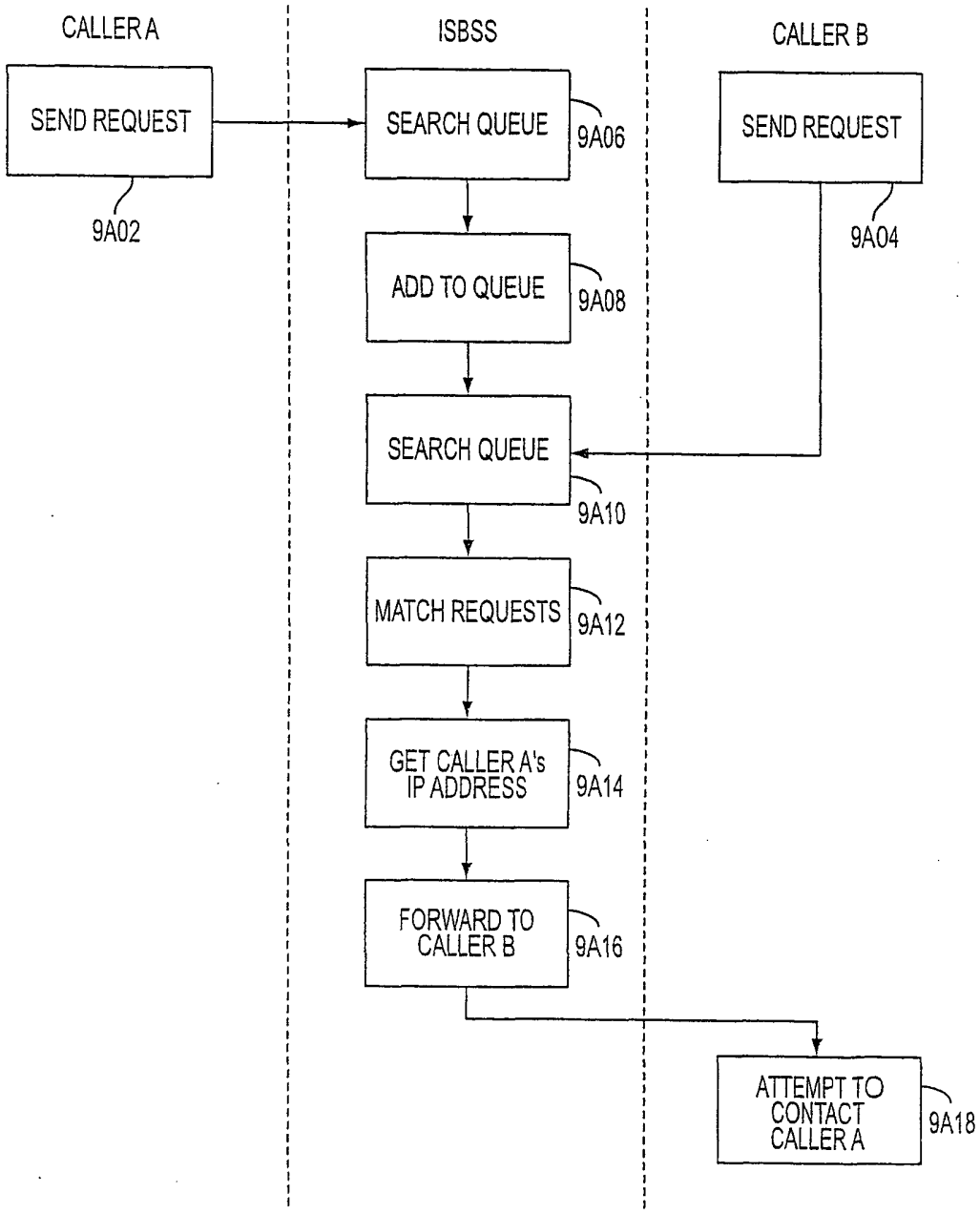
CALLER A
ISBSS
CALLER B
SEND REQUEST
9A02
SEARCH QUEUE
9A06
SEND REQUEST
9A04
ADD TO QUEUE
9A08
SEARCH QUEUE
9A10
MATCH REQUESTS
9A12
GET CALLER A's IP ADDRESS
9A14
FORWARD TO CALLER B
9A16
ATTEMPT TO CONTACT CALLER A
9A18

FIG. 9A

```
unsigned char *itobod (unsigned int decimalNumber, unsigned int &digitCount)
/*
given an integer, produces and returns a BCD (binary-coded
decimal) string, in which each byte (unsigned char) is logically split
into two 4-bit "nibbles", each of which contains one digit of the
original integer.  Also returned as an argument-by-reference is the
number of digits found in the original integer, which is useful for
later manipulations.

The most significant digit of the original integer is stored "first",
i.e. in the high-order nibble of the leftmost byte of the BCD string.

In the current implementation, (non-leading) zeroes in the original
integer are stored as hex digit 'A' (0xA) in order to distinguish them
from "blank" or "filler" nibbles and/or bytes, which actually contain
zeroes.
*/

{
  // these are static to reduce repeat memory allocation-- for FoneFriend

static int numOfBytes;              // bytes needed to store it as BCD

static int numOfDigits;             // for internal use only!
static unsigned char *BCDbuf;       // the return value goes here
static unsigned char *bytePtr;      // moving pointer for loading BCDbuf...
static char BitShift;               // used for decimal-to-hex conversion
static char BCDdigits[10] =           // this allows us to do tricks like
   { 0xA, 1, 2, 3, 4, 5, 6, 7, 8, 9 }; // storing digit 0 as 0xA

  // figure out the number of digits in 'decimalNumber'
numOfDigits = log10((double) decimalNumber) + 1;
if (numOfDigits <= 0)
  return NULL;
digitCount = numOfDigits;  // digitCount is returned to the user

numOfBytes = (int) ceil((double) numOfDigits / 2.0 );

   //set up storage and pointers accordingly
BCDbuf = new unsigned char[numOfBytes];
bytePtr = &BCDbuf[numOfBytes-1];

  // clear out the contents of BCDbuf-- correct functioning depends on this
bzero(BCDbuf, numOfBytes) ;
```

FIG. 10A

```
  // we are storing BCD digits from most to least significant, going
  // left to right; and there are two digits per byte.  If there are
  // an odd number of digits to store, then the least significant decimal
  // digit will wind up in the HIGH-order nibble of the last (rightmost)
  // byte used; if there are an even number of digits, this last digit
  // will end up in the LOW-order nibble of the last byte.  Since we start
  // by storing the least significant decimal digit and move backwards,
  // we have to know right away which nibble to put it in.  QED.
if (numOfDigits % 2)      // we have an odd number of digits
   BitShift = 4;          // start in high-order nibble (left-shift 4 bits)
else BitShift = 0;        // start in low-order nibble (no shift)

 while (numOfDigits--) {    // we have at least one more digit to do

    // get the last digit of 'decimalNumber' and put it in the
    // appropriate nibble
  *bytePtr += (BCDdigits[decimalNumber % 10]  << BitShift);

    // now, we need to get ready to deal with the next digit.
    // crafty code alert! BitShift can have the values 0 and 4; if it
    // is currently 0, then we just handled the LOW-order nibble of a
    // byte, and we will stay within this byte to do the next digit.
    // But if BitShift is currently 4, we just did the HIGH-order byte
    // and we can move back to the previous byte.  The following
    // very confusing code does that for you:
  bytePtr -= (BitShift / 4) ;

    // of course, the value of BitShift must now be toggled:

  BitShift = 4 - BitShift;

    // finally, we line up 'decimalNumber' to deal with the next digit
    // in line, by way of throwing away the last digit we looked at, which
    // was the least significant digit of decimalNumber'.
  decimalNumber  /= 10;

    // at long last, we're ready to copy the digit into the BCD string:
  //    *bytePtr += (BCDdigits[decimalNumber % 10] << BitShift);

 }

  return BCDbuf;
}
```

FIG. 10B

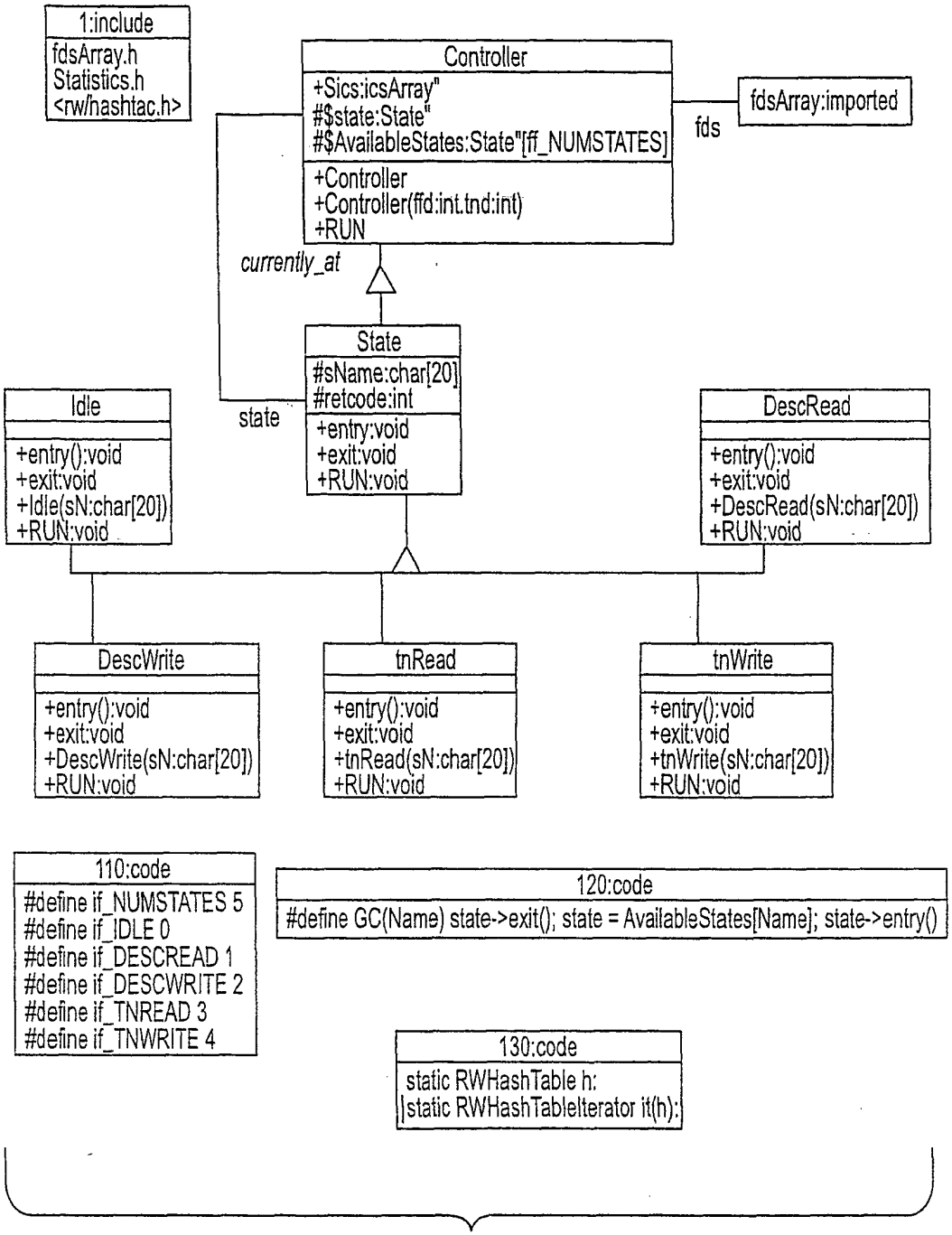
1:include
fdsArray.h
Statistics.h
<rw/hashtac.h>
Controller
+Sics:icsArray"
#$state:State"
#$AvailableStates:State"[ff_NUMSTATES]
+Controller
+Controller(ffd:int.tnd:int)
+RUN
fdsArray:imported
fds
currently_at
State
#sName:char[20]
#retcode:int
+entry:void
+exit:void
+RUN:void
state
Idle
+entry():void
+exit:void
+Idle(sN:char[20])
+RUN:void
DescRead
+entry():void
+exit:void
+DescRead(sN:char[20])
+RUN:void
DescWrite
+entry():void
+exit:void
+DescWrite(sN:char[20])
+RUN:void
tnRead
+entry():void
+exit:void
+tnRead(sN:char[20])
+RUN:void
tnWrite
+entry():void
+exit:void
+tnWrite(sN:char[20])
+RUN:void
110:code
#define if_NUMSTATES 5
#define if_IDLE 0
#define if_DESCREAD 1
#define if_DESCWRITE 2
#define if_TNREAD 3
#define if_TNWRITE 4
120:code
#define GC(Name) state->exit(); state = AvailableStates[Name]; state->entry()
130:code
static RWHashTable h:
static RWHashTableIterator it(h):

FIG. 11

```
typedef struct {
unsigned short struct_type;
   // tells us how to interpret the tx_data
   // 1001  t_ConnectPacket
   // 1002  t_RxConnectPacket
unsigned short len;  // length of data in tx_data
char tx_data[252];   // 262 bytes to handle future expansion
} tx_packet;
```

FIG. 11A

```
typedef struct {
     unsigned char hw_version; // identifies the originator of this struct
   unsigned char sw_version; // 1 == 1st version

     //  the connection type should be the first bytes read.
     //  the types are:
     //       1 - caller non-1st time
     //       2 - called
     //       3 - caller 1st time
     //       4 - mmic
     //       5 - message
     //       7 - self-test
     //       8 - upgrade request
   unsigned short int connect_type;
   unsigned char my_phone_num[8];
   unsigned char his_phone_num[8]:
   unsigned long my_serial_num;
   unsigned long his_serial_num;
   unsigned char my_ip[4];
   t_BillingData bill_rec;
} t_ConnectPacket;
```

FIG. 11B

```
typedef struct {
      unsigned long start_time; // start time of previous service
      unsigned long stop_time;  // duration (in seconds) of previous service
      unsigned char phone[8];   // phone number of previous call

      unsigned char stat_data[8]; // statistical data about previous service
  } t_BillingData;
```

FIG. 11C

```
typedef struct {
  unsigned short struct_type;
     // tells us how to interpret the tx_data
     // 1001  t_ConnectPacket
     // 1002  t_RxConnectPacket
  unsigned short len;  // length of data in tx_data
  char tx-data[252];   // 252 bytes to handle future expansion
} tx packet;
```

FIG. 11D

```
typedef struct {
  // New fields added to allow for commands
  unsigned char pckt_type;  // 0 == message, 1 == error
  unsigned char me_type;
     // messages:
     // 0 = return usable IP addr,
     // 1 = no match: IP == 0.0.0.0,
     // 2 = go to another server; IP address given
     // 3 = no action to take (response to message or self-test; IP == 0.0.0.0)
     // errors:
     // 0 = problem on my end; retry from scratch
     // 1 = problem with your data; retry from scratch
     // 2 = you are not an active user of the requested FF Service.
  unsigned char commandType;
     // 0 == no command
     // 1 == contact command server for further commands
     //        send new IP addr in command
     // 2 == set Update Available light on
     // 3 == unset Update Available light
     // 4    new main server
     //        send new IP addr in command
     // 5 == new backup sever
     //         send new IP addr in command
  unsigned char commandSize;   // number of bytes found in command []
  unsigned char his_ ip [4];
  unsigned long cur_time;
  char command[32];
     // If commandSize <= 28 we can rely on
     // bytes command[28] .. command[31] containing the
     // sender serial number just for debugging purposes.
     // we have not specified what a command looks like.
     // commandType == 2:
     //   commandSize = 8, command = "10 2 1\r\n"
     // commandType == 4:
     //   commandSize = 21, command = "0 1 0 137 140 7 222\r\n"
     // commandType == 21:
     //   commandSize = 8, command = "0 1 1 137 140 7 222\r\n"
} t_RxConnectPacket;
```

FIG. 11E

```
********** Results from generation of Statistics **********

***** Absolute Value Counters *****
  m Entered Idle state          : 985131
  m FFServer connection Requests: 0
  m Entered DescRead state      : 0
  m Entered DescWrite state     : 0
  m DescRead  ok                : 0
  m DescRead failed: wrong size : 0
  m DescRead failed: disconnect : 0
  m DescRead failed: orderly rel: 0
  m DescWrite ok                : 0
  m DescWrite failed            : 0
  m Init New Descriptor         : 1
  m Conn discon in complete list: 0
  m Invalid Client Port         : 0
  m Entered Housekeeping        : 985099
  m Completed Connection RQ     : 0
  m Expired Connection RQ       : 0
  m Inactive Connection RQ      : 0
  m tnClient Write ok           : 29
  m tnClient Write failed       : 0
  m Serial Number Invalid       : 0
***** Maximum Value Counters *****
  m Max Complete Connection Q   : 0
  m Max Stack Size              : 0
  m Max Connection List Size    : 0
***** Minimum Value Counters *****
  m Min Stack Size              : 2147483647
  m Min Connection List Size    : 0
********** End of StatisticsReport **********

Monitoring Stopped
```

FIG. 11F

```
Mon Feb 23 13:06:31 1998> New logged session of FFServer

Mon Feb 23 13:06:31 1998> Number of Invalid Serial Numbers: 1000
Mon Feb 23 13:06:55 1998> New TNClient (IP.Port): 137.140.8.104.36239
Mon Feb 23 13:07:55 1998> Closing TNClient (IP.Port) = 137.140.8.104.36239

Mon Feb 23 13:07:56 1998> (CL) Unknown ConnectType (IP.Port): 137.140.8.104.36239
Mon Feb 23 13:07:57 1998> (CL) Wrong Packet Size  (IP.Port): 137.140.8.104.36239
Mon Feb 23 13:07:58 1998> (CL) PcktType 1= 1001  (IP.Port): 137.140.8.104.36239
Mon Feb 23 13:07:59 1998> (CL) tx_packetPtr was NULL (IP.Port): 137.140.8.104.36239
Mon Feb 23 13:07:60 1998> (CL) Failed on attempt to insert  (IP.Port): 137.140.8.104.36239
```

FIG. 11G

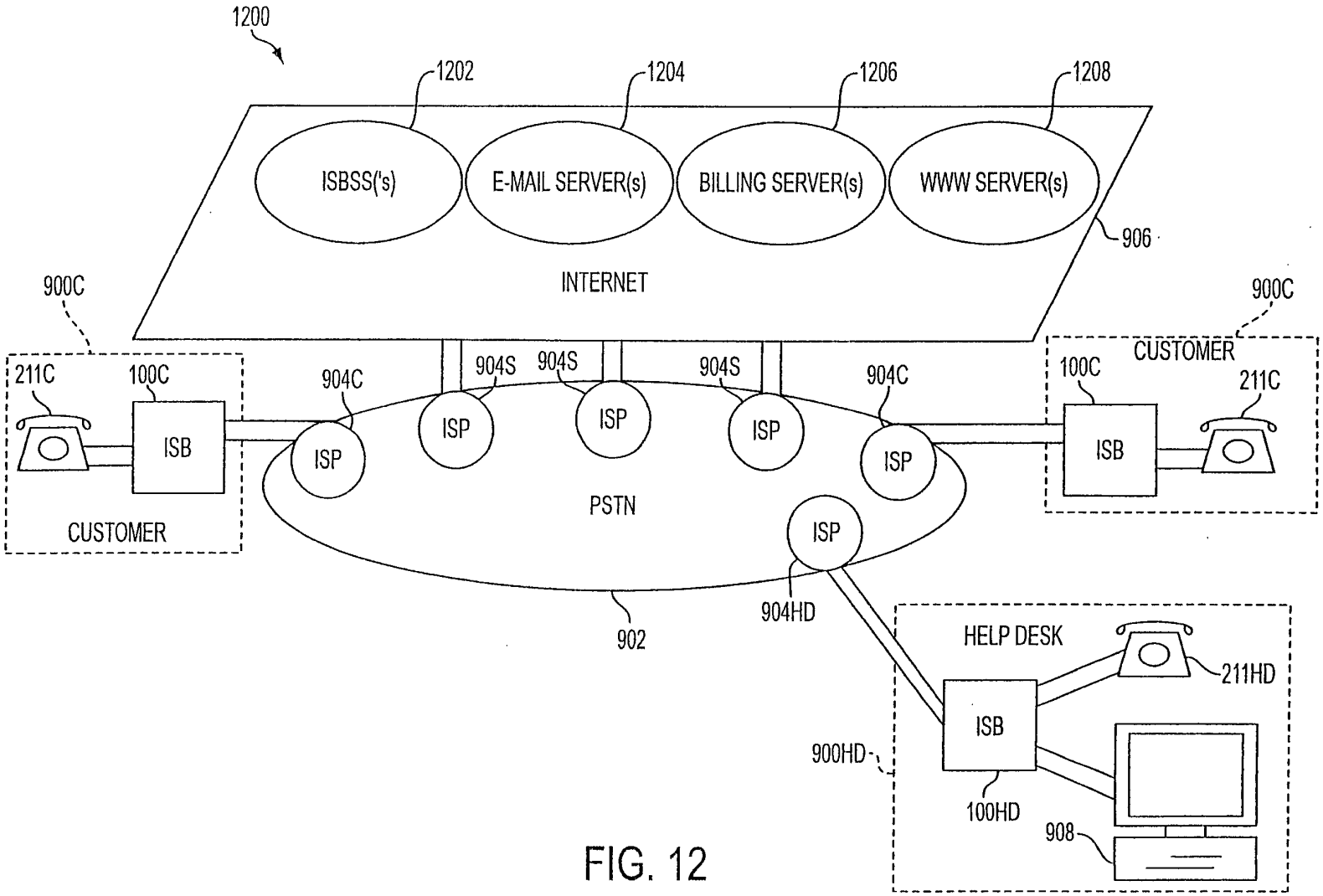
1200
1202
1204
1206
1208
ISBSS('s)
E-MAIL SERVER(s)
BILLING SERVER(s)
WWW SERVER(s)
906
INTERNET
900C
211C
100C
ISB
CUSTOMER
904C
904S
904S
904S
904C
ISP
ISP
ISP
ISP
ISP
ISP
PSTN
902
904HD
900C
CUSTOMER
100C
211C
ISB
HELP DESK
211HD
ISB
900HD
100HD
908

FIG. 12

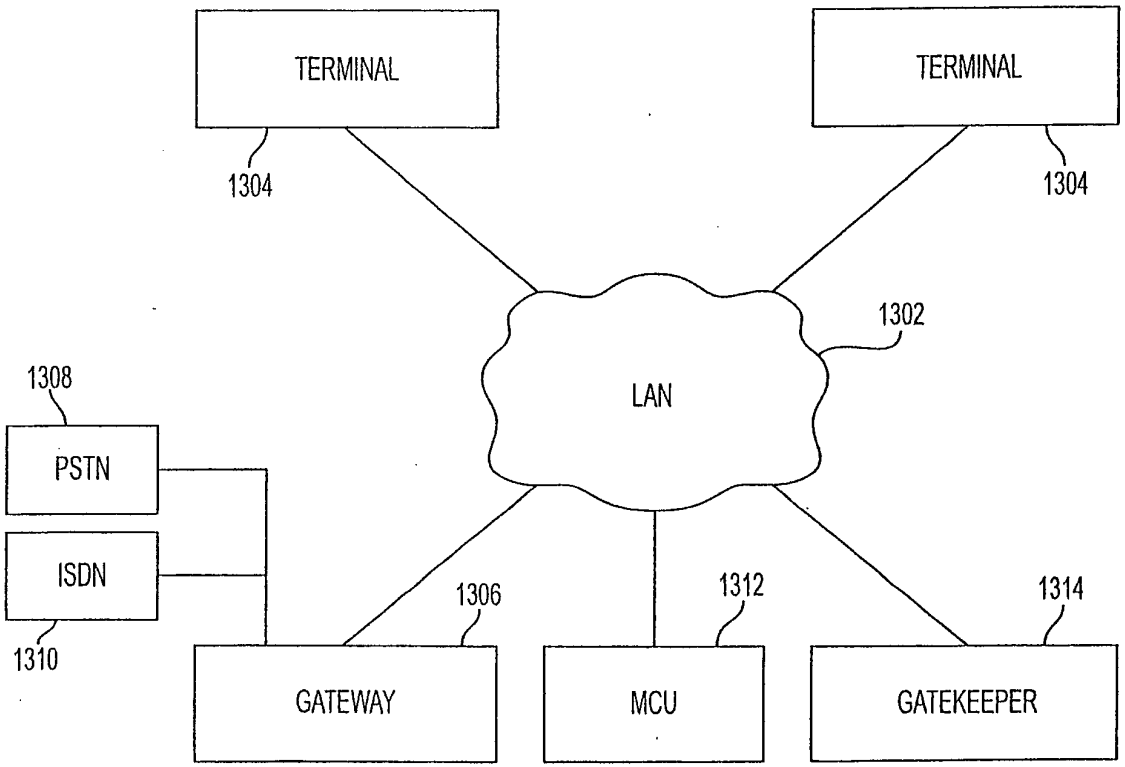
TERMINAL
TERMINAL
1304
1304
1302
LAN
1308
PSTN
ISDN
1310
1306
1312
1314
GATEWAY
MCU
GATEKEEPER

FIG. 13